Universidad de Costa Rica

Escuela de Matemática

Herramientas de Ciencias de Datos I (CA-0204)

Bitacora 3

Bot Hatchet

Profesor: Luis Juárez Potoy

Estudiantes:
Andres Zuñiga Mora (C38733)
Anthonny Flores Rojas (C32975)
Amy Chen Wu (C32203)
Leonardo Vega Aragon (C38313)

Fecha: 2025-11-23

Índice general

1. Escoja técnicas o métodos estadísticos que le permitan responder su pregunta de investigación.

Nuestra Pregunta de Investigación es: ¿Qué tan factible es hacer un bot de ajedrez en un lenguaje como R?

Técnicas o Métodos Estadísticos Escogidos:

- **Porcentaje de victorias (Win Rate):** Permite medir el desempeño general del bot en función del número de partidas ganadas respecto al total.
- **Promedio de tiempo por movimiento:** Evalúa la velocidad de respuesta del bot, lo cual es relevante para determinar si el funcionamiento es eficiente en tiempo real.
- **Promedio de jugadas por partida:** Permite identificar la duración de las partidas y si el bot logra sostener juegos completos y válidos.
- **Análisis de tendencia del aprendizaje:** Se calcula el cambio del porcentaje de victorias a lo largo de múltiples partidas para determinar si el rendimiento mejora con la experiencia.

Estas técnicas permiten analizar objetivamente si un bot de ajedrez implementado en R es capaz de competir de forma estable y eficiente frente a un motor profesional como Stockfish, lo cual responde directamente a la pregunta de investigación.

2. Escriba un marco metodológico donde se coloque la fundamentaciónn estadístico-matemática del método o técnica escogida.

1. Modelado del Problema: Proceso de Decisión de Markov (MDP)

Para modelar las patidas de ajedrez de la manera más óptima posible, se decidió utilizar el **Proceso de Decisión de Markov** (MDP) [8]. Este proceso se desarrolla en un entorno

de juego con información perfecta y cuenta con tres componentes:

- **Estado (s)**:El estado corresponde a una descripción completa y precisa de la posición actual del juego. Así, se modeló el tablero como un tensor de dimensiones $8\times8\times16$. Esta representación codifica, para cada casilla, la presencia o ausencia de cada tipo de pieza (propia y del oponente), así como otros atributos relevantes del estado del juego [4]. Esta estructura permite que el modelo procese la información del tablero de manera uniforme y facilita su uso como entrada para algoritmos de aprendizaje profundo. Un estado también puede incluir información adicional como el turno, el derecho de enroque o la posibilidad de captura al paso, pero para este proyecto se mantuvo una representación simplificada centrada en las piezas activas.
- **Acción (a)**: Una acción corresponde a cualquier movimiento legal que un jugador puede ejecutar desde la posición actual del tablero. Para esto, se usó la lista de movimientos permitidos en el juego del ajedrez.
- **Recompensa (R)**: La recompensa utilizada fue terminal y binaria, representada por

$$z \in \{-1,\, 0,\, 1\}.$$

  Esto significa que el agente solo recibe una recompensa al finalizar la partida:
  - $z = 1$ si el bot gana,
  - $z = -1$ si pierde,
  - $z = 0$ si empata.

  Este diseño simplificado evita sesgos durante el aprendizaje y obliga al modelo a identificar patrones que conducen a posiciones ventajosas a largo plazo [8]. No se asignaron recompensas intermedias para evitar un aprendizaje miope basado en valores heurísticos y, en su lugar, se buscó que el bot aprendiera estrategias coherentes a lo largo del horizonte completo de la partida.
- **Objetivo**: Encontrar la política óptima $\pi^*(a \mid s)$ y la función de valor óptimo $V^*(s)$.

El proyecto se fundamenta en el marco del **Aprendizaje por Refuerzo Profundo (Deep Reinforcement Learning - DRL)** [7], empleando una arquitectura tipo AlphaZero que combina una **Red Neuronal de Valor y Política** con el algoritmo de **Búsqueda en**

Árbol Monte Carlo (MCTS) [8].

2. Red Neuronal Profunda: Aproximación de Funciones

La red neuronal $\mathbf{f}_\theta(s)$ aproxima el valor y la política utilizando dos cabezas de salida [5], [6].

A. Cabeza de Valor (Value Head)

Aproxima la función de valor $V^*(s)$, es decir, la esperanza de la recompensa terminal. La activación final es $\tanh(\cdot)$.

Aproximación:

$$\hat{v}_\theta(s) \approx V^*(s)$$

Función de Pérdida (Error Cuadrático Medio - MSE):

$$\mathcal{L}_v(\theta) = \frac{1}{N}\sum_{i=1}^{N}(z_i - \hat{v}_\theta(s_i))^2$$

Donde z_i es el resultado final real de la partida (recompensa).

B. Cabeza de Política (Policy Head)

Aproxima la distribución de probabilidad óptima $\pi^*(s)$ sobre las acciones. La activación final es $\text{softmax}(\cdot)$.

Aproximación:

$$\hat{p}_\theta(a \mid s) \approx \pi^*(a \mid s)$$

Función de Pérdida (Entropía Cruzada Categórica - CCE): Minimiza la divergencia entre la política predicha $\hat{p}_\theta$ y la política objetivo π generada por MCTS [7], [8].

$$\mathcal{L}_\pi(\theta) = -\sum_{a \in \text{Acciones}} \pi(a \mid s) \cdot \log(\hat{p}_\theta(a \mid s))$$

C. Pérdida Total

La función de pérdida total combina ambas cabezas con un término de regularización L_2 (λ es el coeficiente de regularización) [8].

$$\mathcal{L}(\theta) = \mathcal{L}_v(\theta) + \mathcal{L}_\pi(\theta) + \lambda||\theta||^2$$

El entrenamiento se realiza por **Descenso de Gradiente** (Optimizador Adam) para encontrar $\text{argmin}_\theta \mathcal{L}(\theta)$.

3. Algoritmo de Búsqueda en Árbol Monte Carlo (MCTS)

El MCTS refina las predicciones de la red neuronal mediante simulación, generando una política de búsqueda π superior a la política a priori $\hat{p}$ [8].

A. Criterio de Selección (PUCT)

El MCTS selecciona la acción a que maximiza el valor $U(s, a)$, equilibrando la explotación de las acciones con alto Q y la exploración de las acciones con alta probabilidad $\hat{p}$ o baja visita $N(s, a)$ [8].

$$U(s, a) = Q(s, a) + C_{\text{PUCT}} \cdot \hat{p}(a \mid s) \cdot \frac{\sqrt{N(s)}}{1 + N(s, a)}$$

B. Generación de la Política Objetivo

Tras las simulaciones, la política $\pi(a \mid s)$ usada como **etiqueta de entrenamiento** se define como la distribución de probabilidad basada en las frecuencias de visita $N(s, a)$.

$$\pi(a \mid s) = \frac{N(s, a)}{\sum_b N(s, b)}$$

C. Muestreo con Temperatura

La selección final del movimiento en el auto-juego se realiza aplicando una temperatura τ a la política de búsqueda, permitiendo la transición de **exploración** ($\tau = 1.0$ al inicio) a

explotación ($\tau = 0.01$ al final) mediante:

$$\text{Prob}(a) = \frac{\pi(a \mid s)^{1/\tau}}{\sum_b \pi(b \mid s)^{1/\tau}}$$

3. Aplique dichas técnicas o métodos. Presente los resultados, analícelos y responda su pregunta de investigación.

Pregunta de investigación

¿Es factible desarrollar un bot de ajedrez en un lenguaje como R?

Evidencia basada en los datos recolectados

A partir de los registros almacenados en `data_fuerte.csv`, se observa lo siguiente:

- Todas las partidas fueron ejecutadas en el modo `bot_vs_bot_interno`, es decir, la IA desarrollada en R jugó contra sí misma.
- Todas las partidas finalizaron correctamente, sin errores del motor de ajedrez o fallos durante la ejecución.
- La variable `Resultado_Num` registra únicamente valores iguales a 0, lo que indica que todas las partidas terminaron en **empate**.
- Las variables `Mate` y `Jaque` permanecen como `FALSE`, evidenciando que las partidas no concluyen mediante jaque mate, sino por mecanismos de **tablas** (por estancamiento, repetición o agotamiento de recursos para progresar).
- El número de movimientos por partida se encuentra entre aproximadamente 50 y 90 jugadas, lo que evidencia que la IA es capaz de mantener partidas completas y estables.
- El tiempo promedio por movimiento se encuentra alrededor de **0.04 a 0.05 segundos**, lo cual se considera un rendimiento alto en términos de eficiencia de ejecución.

Además, durante las ejecuciones del modo IA vs Stockfish, se observó que la IA desarrollada en R:

1. Es capaz de producir movimientos legales y sostener una partida completa.
2. No presenta fallos informáticos ni errores del motor.
3. No posee capacidades estratégicas profundas ni algoritmos de búsqueda avanzados.
4. Pierde consistentemente ante Stockfish debido a la ausencia de evaluación posicional y de planificación táctica.

Interpretación

Los resultados muestran que el bot implementado en R logra jugar de forma completa, estable y eficiente. Esto confirma que, desde un punto de vista computacional, es **posible desarrollar un bot de ajedrez funcional** dentro de este lenguaje. Sin embargo, el nivel de competitividad del bot es bajo. La IA no evalúa posiciones, no calcula variantes profundas ni implementa principios estratégicos de apertura, medio juego o finales. Como consecuencia, el bot comete pérdidas de material sin compensación y no puede identificar amenazas tácticas relevantes, lo cual explica que pierda de manera consistente ante Stockfish.

Respuesta a la pregunta de investigación

A partir de los resultados obtenidos, se concluye que **sí es factible desarrollar un bot de ajedrez en un lenguaje como R**, en el sentido de que el sistema puede generar movimientos legales, completar partidas completas sin fallos, comunicarse correctamente con un motor profesional como Stockfish y registrar información relevante para su análisis posterior.

No obstante, los resultados también indican que **R no es un lenguaje adecuado para alcanzar un nivel competitivo alto** en el contexto de motores de ajedrez. Esto se debe a que R no está optimizado para cálculos intensivos en búsqueda de árboles de decisión, heurísticas posicionales, paralelización específica para análisis de ajedrez ni administración de memoria de bajo nivel, aspectos fundamentales en los motores modernos. Lenguajes como C++, Rust o incluso Python con bibliotecas especializadas ofrecen herramientas significativamente más eficientes para este tipo de aplicaciones [11].

Síntesis

El bot desarrollado demuestra que R es técnicamente capaz de:

- procesar posiciones de ajedrez y generar jugadas válidas,
- interactuar con un motor profesional en tiempo real,
- mantener partidas completas sin errores,
- y almacenar datos para investigación y análisis estadístico.

Sin embargo, **no es capaz de competir eficazmente contra motores avanzados debido a sus limitaciones de rendimiento y capacidad algorítmica en este contexto**. Por tanto, R resulta útil para fines educativos, experimentales y analíticos, pero no para el desarrollo de un motor de ajedrez de alto nivel.

6. Escriba tres conclusiones obtenidas de su trabajo dando respuesta su pregunta de investigación.

Conclusiones

La primera conclusión a la que se llegó durante el desarrollo del presente proyecto fue que todavía hace falta una gran cantidad de aprendizaje para lograr usar de manera correcta el concepto de redes neuronales. Una de las mayores dificultades se dio en que, cada vez que se quería hacer un avance había que estudiar una serie de conceptos, los cuales a su vez tenían raíces en otros temas aparte. En síntesis, este tema es sumamente amplio, y en el transcurso de un semestre resulta sumamente difícil aprender a manejar la teoría detrás de las redes neuronales.

Posteriormente, la segunda conclusión que se obtuvo es que el entrenamiento de un bot en el contexto del ajedrez es sumamente eficiente, pues con una cantidad relativamente baja de entrenamientos el bot fue capaz de alcanzar el nivel de un jugador promedio de ajedrez. Esto último fue gracias al estudio efectivo del funcionamiento de Elo y del criterio de las jugadas que se realizan durante una partida. Todo lo anterior muestra el potencial tan grande

y efectivo que tiene este tipo de entrenamientos, y abre la puerta a poder generalizar las ideas usadas en el transcurso del proyecto a otra área en donde sea necesario entrenar a una máquina.

Finalmente, el proceso de implementar el motor de ajedrez de Stockfish en el proyecto fue un proceso que resultó tomar más tiempo del que se tenía previsto desde un inicio, pues se llegaron a presentar una gran serie de errores que provocaron la necesidad de estudiar a fondo el funcionamiento del motor. Sin embargo, los resultados fueron satisfactorios, aunque el segmento de código asociado a Stockfish terminó siendo bastante pesado en términos de memoria.

A nivel general, este proyecto resultó ser un gran reto para los miembros del grupo, con varios aprendizajes de por medio. Sin embargo, por las dificultades asociadas al uso de redes neuronales fue que el desarrollo de este proyecto se vio limitado en gran medida.

7. Escriba las limitaciones que tuvo al desarrollar el proyecto.

El desarrollo del proyecto, si bien ambicioso al emular arquitecturas de inteligencia artificial como AlphaZero, enfrentó limitaciones significativas tanto en el ámbito de la experiencia técnica, teorica, y la programación como en el de los recursos computacionales.

Limitaciones Académicas y Técnicas

La principal barrera fue el bajo background teórico del equipo en la creación y el entrenamiento de Redes Neuronales Profundas [5], [6] y Aprendizaje por Refuerzo [8]. Esto obligó al equipo a una extensa y rápida curva de aprendizaje para asimilar modelos complejos ya existentes, como la arquitectura ResNet de doble cabeza utilizada en AlphaZero, lo que consumió un tiempo considerable que habría sido dedicado a la optimización y pruebas del modelo. Sumado a esto, las diferencias en la experiencia en programación en R [11] entre los participantes introdujo problemas de eficiencia en la codificación, particularmente en la

implementación de algoritmos complejos como la Búsqueda en Árbol Monte Carlo (MCTS) y la gestión eficiente de los tensores de datos (la función `fen.to.vector`).

Limitaciones Computacionales y de Escala

La limitación más crítica fue la capacidad de cómputo. Los modelos de DRL para juegos como el ajedrez requieren una gran cantidad de auto-entrenamiento para converger a una política superior [8]. Para obtener resultados concluyentes y una mejora significativa, se estima que se debería haber jugado al menos 1,000 partidas o más, un volumen que a pesar que se hizo uso de GPU especializa, es inviable sin acceso a clústeres de GPUs de alto rendimiento por parte de todos los colaborados. Esta restricción impidió realizar la cantidad necesaria de iteraciones de entrenamiento y exploración (`bot.vs.bot`), lo que se tradujo en un tiempo de convergencia extremadamente prolongado y limitó la capacidad del modelo para alcanzar un rendimiento competitivo. En la práctica, esta insuficiencia de datos de entrenamiento limitó la robustez de la política π y la precisión de la función de valor $\widehat{v}$.

Referencias

[1] D. Klein, "Neural networks for chess: The magic of deep and reinforcement learning revealed," arXiv, 2022. [En línea]. Disponible en: https://arxiv.org/abs/2209.01506

[2] C. C. Aggarwal, *Neural networks and deep learning: A textbook*. Cham, Switzerland: Springer, 2018.

[3] C. M. Bishop, *Pattern recognition and machine learning*. New York, NY, USA: Springer Science+Business Media, 2006.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge, MA, USA: MIT Press, 2016.

[5] D. Silver et al., "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018. [En línea]. Disponible en: https://doi.org/10.1126/science.aar6404

[6] S. Alam, "An investigation of the imputation techniques for missing data," *ScienceDirect*, 2023. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S2772662223001819

[7] M. DeLeo and E. Guven, "Learning chess with language models and transformers," Whiting School of Engineering, Johns Hopkins University. [En línea].

[8] R Core Team, "CRAN: Manuals – The Comprehensive R Archive Network." [En línea]. Disponible en: https://cran.r-project.org/manuals.html

Anexo